Tracking Application Fingerprint in a Trustless Cloud Environment for Sabotage Detection

Jean-Emile Dartois, Jalil Boukhobza, Vincent Francoise, Olivier Barais

Tracking Application Fingerprint in a Trustless Cloud Environment for Sabotage Detection

Jean-Emile Dartois*†, Jalil Boukhobza*‡, Vincent Françoise*, and Olivier Barais*†
*b<>com Institute of Research and Technology, †Univ Rennes, Inria, CNRS, IRISA, ‡Univ. Bretagne Occidentale
Email: jean-emile.dartois@b-com.com, boukhobza@univ-brest.fr, vincent.francoise@b-com.com, barais@irisa.fr

***Abstract*—Companies are more and more inclined to use collaborative cloud resources when their maximum internal capacities are reached in order to minimize their TCO. The downside of using such a collaborative cloud, made of private clouds' unused resources, is that malicious resource providers may sabotage the correct execution of third-party-owned applications due to its uncontrolled nature. In this paper, we propose an approach that allows sabotage detection in a trustless environment. To do so, we designed a mechanism that (1) builds an application fingerprint considering a large set of resources usage (such as CPU, I/O, memory) in a trusted environment using random forest algorithm, and (2) an online remote fingerprint recognizer that monitors application execution and that makes it possible to detect unexpected application behavior. Our approach has been tested by building the fingerprint of 5 applications on trusted machines. When running these applications on untrusted machines (with either homogeneous, heterogeneous or unspecified hardware from the one that was used to build the model), the fingerprint recognizer was able to ascertain whether the execution of the application is correct or not with a median accuracy of about 98% for heterogeneous hardware and about 40% for the unspecified one.**

***Keywords*-Correct Execution, Resource Management, Cloud, Fingerprint, Sabotage**

I. Introduction

Companies are using more and more computing resources for processing their data and for providing the best Quality of Service (QoS) possible to their customers [1]. These computing resources have a significant cost and companies are seeking solutions to reduce their Total Cost of Ownership (TCO) without sacrificing the correct execution of their applications.

A promising alternative for optimizing the cost of processing applications on Cloud infrastructures is to opportunistically exploit their allocated but momentarily unused computing resources [2]. Many platforms (*e.g.,* BOINC [3], Condor [4]) enable the leveraging of these unused resources for a variety of purposes (*e.g.,* scientific computing, big data) and business models (*e.g.,* free, reward). However, any infrastructure owner (*i.e, Farmer*) can join such platforms to provide/share his computation capacities. These farmers seek to reduce their TCO by making their unused computing resources available to other users. Allowing any farmer to join such platforms exposes an *Operator* (*i.e,* interface organizations between the farmers and the customers) to malicious behavior. Malicious farmers can potentially produce erroneous or inaccurate results without effectively running the applications to obtain higher benefits from the Operator (*e.g.,* while saving their computation capacities) [5]. In such a scenario, one needs to investigate *how can an operator prevent malicious infrastructure owners from sabotaging the remote computation of its customers.*

Many studies have been conducted to provide secure remote computation [6], [7]. Most of the traditional approaches such as replication voting, ringers, and spot checking - whether with or without blacklisting - have a high overhead on the compute resources (it may double the used resources) to verify each application execution or requires a dedicated hardware such as Intel SGX with 60% of the native throughput and about 2x increase of the application code size [6].

In this paper, we propose a different but complementary solution to state-of-the-art work having the following properties:

- **Backward compatibility**: non-invasive/non-intrusive on the application code and not limited to a type of application or hardware.
- **Online execution**: continuous verification of the correct execution of the application.
- **Efficiency**: proving a small overhead to verify each application execution

Our approach relies on the use of classification techniques to build a fingerprint model of an application execution in a trusted environment using the Random Forest learning algorithm. In this work, we assume that performance metrics are continuously sent by the farmer as in [8], [9]. Then, the built trusted fingerprint model is continuously compared with the current workload metrics sent from the untrusted environment to detect an application execution sabotaging or the alteration of its behavior. To do so, three different cases can be observed:

- The **homogeneous hardware** case where the targeted hardware is both standardized and specified. This means here that the model is trained with this standardized hardware as the targeted one.
- The **heterogeneous hardware** case where the targeted

hardware is specified but varies from machine to machine. This means that the model is trained with the same (heterogeneous) hardware mix as the targeted one.
- The **unspecified hardware** case where the targeted hardware is both unspecified and heterogeneous. This means that the model is not trained with the same hardware mix as the targeted one.

We have investigated five applications: multimedia processing, file server, 3D rendering, software development, web application. The used dataset represents about 10 hours of applications execution on the four tested physical heterogeneous machines.

Our experimental results show that our fingerprint recognizer is able to detect the correct execution of applications in a trustless environment with a median accuracy of 99.88% for *homogeneous hardware*, 98% when using *heterogeneous hardware* and 44% in case of *unspecified hardware* during the training phase (see Section IV).

The remainder of the paper is organized as follows. Section II presents some background knowledge. Then, our methodology is described in Section III. Section IV details the experimental evaluation performed. Section VI reviews related work. Finally, we conclude in Section VII.

II. Background

This section gives some background on machine learning.

Machine learning investigates automatic techniques to make accurate forecasts based on past observations [10]. Datasets contain a set of parameters called features, used to build a forecasting model for some specific output response metrics. Datasets can either be quantitative (e.g., throughput) or categorical (e.g., spam/not spam). Metrics such as CPU, memory, I/O, and network resources are examples of features while the application name is the label.

There are three different categories in machine learning: supervised, unsupervised and reinforcement learning. Our objective in this study is to build a forecasting model that detects existing relationships between features (*e.g.,* CPU usage, dirty page, I/O read, active file) and a label (the running application). As a consequence, our problem fits in the supervised learning category. Since we want to predict the running remote application which is a categorical value, we used classification-based algorithms.

III. Methodology

In order to monitor application resource usage, different metrics (*e.g.,* CPU usage, memory usage, throughput, etc.) are utilized. Analyzing and characterizing these metrics would enable one to create predictive fingerprint recognition models that make it possible to verify that the remote machines are effectively executing the requested applications. One assumption we made is that the farmers are providing those resource usage metrics online for the container used to execute the customer application.

To create such predictive fingerprint recognition models, we propose a framework that is able to control the correct execution of applications in a trustless environment. Our framework is made of three components (see Figure 1). This paper focuses on the *Fingerprint Tracker* and *Fingerprint Builder*.

1) The **Decision Engine** is responsible for handling customer requests which consist in executing the designated applications (1). To do so, the decision engine first verifies whether the fingerprint recognition model for the requested application is available. If not, it requests the *Fingerprint Builder* to generate one for this new application (2). Then, the Decision Engine chooses a suitable farmer that will be in charge of executing the customer application (3) [11]. Finally, it requests the *Fingerprint Tracker* to verify the correct execution of this application (4).
2) The **Fingerprint Builder** is responsible for constructing the predictive fingerprint recognition models. To do so, this component uses an environment of trust in order to ascertain the correctness of such models (see Section III-A).
3) The **Fingerprint Tracker** is in charge of controlling continuously the correct execution of applications in a trustless environment (*i.e.,* the farmer infrastructure) using the predictive fingerprint recognition models previously built by the *Fingerprint Builder*. In order to achieve that, it first collects the required execution metrics (5). Then, it identifies the application based on its fingerprint obtained via its resource usage. Finally, it compares this result with the expected application that was communicated by the *Decision Engine* to determine whether or not the application was correctly executed to trigger potential counter measurements when necessary (6) (see Section III-B).

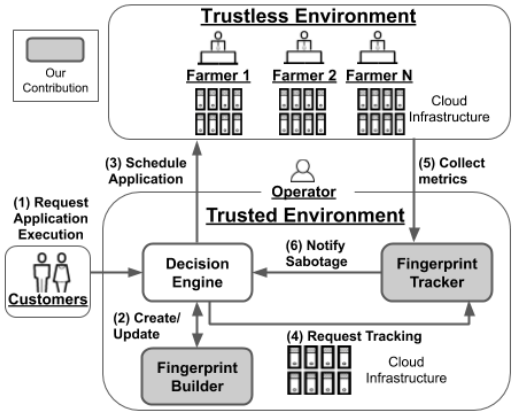

Figure 1. Overall Approach

A. Fingerprint Builder: Building the fingerprint models in an environment of trust

This section details how the *Fingerprint Builder* constructs the fingerprint recognizer models. Figure 2 describes the overall approach followed with three different steps performed in the trusted environment: Data generation step, Learning step, and Evaluation step.

1) Data Generation step: In the dataset generation phase, we have mainly two steps (see Figure 2): generating the traces by executing different applications and collecting their respective container metrics. We have selected five applications that were deployed in a container-based environment covering various use-cases. Table I summarizes the benchmarks used.

Table I
APPLICATIONS AND BENCHMARKS USED

Name	Category	Description
web	Server application	N-tiers web application
email	Server application	Email server
video	Multimedia processing	H.264 video transcoding
rendering	Multimedia processing	3D rendering
compilation	Software build	Linux kernel compilation

To generate the dataset, we used the tools Nginx [12], MySQL [13], and WordPress [14] for the *web* application, FileBench [15] for the *email*, ffmpeg [16] for the *video* application, and blender [1] for the *rendering* application, GNU Compiler Collection [17] for the *compilation* application.

Server application: We chose two typical enterprise server applications: an n-tiers **web application** (WordPress), and email servers (Filebench). WordPress is an Open Source content management system. In our setup, WordPress is deployed with Nginx, PHP, and MySQL. In the case of a WordPress website, we varied the number of concurrent readers/writers between 1 and 50. Varying the number of users has a direct impact on resource usages. The tool that generates the traffic was executed on a separate host. We used Filebench to evaluate **email** to generate a mix of open/read/write/close/delete operations.

Multimedia processing: ffmpeg is a framework dedicated to audio and **video processing**. We used two videos, a FullHD (6.3 GB) and an HD (580MB) video. For the transcoding of the H.264 video, we varied the PRESET parameter between *slow* and *ultrafast*. This parameter has a direct impact on the quality of the compression as well as on the file size. Blender is a toolset for making **3D rendering**, visual effects, art, and interactive 3D applications. We used five 3D models.

Software build: Linux kernel compilation uses thousands of small source files. Its compilation demands intensive CPU usage and short intensive random I/O operations to read a large number of source files and write the object files to the disk. For the sake of our study, we compiled the Linux kernel 4.2.

[1] blender.org

2) Learning step: Choosing the right learning algorithm for a specific problem is a challenging issue. Many state-of-the-art studies such as [18] have discussed the way to select the appropriate learning algorithm(s) depending on the datasets and the type of problem to solve. The accuracy of a model strongly depends on the dataset and the used learning algorithm. In our case, we used Random Forests (RF) to recognize application fingerprints. RF was introduced in [19], the authors enhanced decision trees by building a large collection of de-correlated trees, and then averaging them. RF are a combination of CART (Classification and Regression Trees) models, which are binary trees, such that each model depends on the values of a random vector sampled from training data independently with the same distribution for all trees in the forest. In *CART*, the split aims to maximize the accuracy score by splitting the training data with the best feature on each node of the tree. RF accuracy depends on the tuning parameters, called hyperparameters. These hyperparameters impact the complexity of the learning model, and they are estimated to minimize the error.

3) Data Pre-processing: The goal of the pre-processing step is to create the matrix of input features noted x and the vector of the observed labels noted y (*i.e.,* the running application) from the traces stored in the time-series database. The selection of the input features x is a key step to build a good predictive fingerprint model. One needs to consider the variables that have an influence on application fingerprints for the learning algorithms to find the (hidden) relationships between x and y (see Section II).

4) Feature Extraction: In a container environment, there are more than 50 collected metrics such as active files, CPU usage, I/O async, mapped file, pgpgout. This large number of potential features does not allow for an exhaustive search [20]. According to [21] a good feature selection algorithm can be used based on the following considerations: simplicity, stability, number of reduced features, classification accuracy, storage, and computational requirements. According to [22], PTA(l,r), GPTA(l,r), Sequential Feature Selection (SFFS), and genetic algorithm perform well for such a task.

We used a genetic algorithm to derive a combination of features that maximizes the accuracy of the fingerprint recognition model. Genetic Algorithms (GA) are stochastic optimization methods that mimic the process of natural evolution [23]. In GA, a population is composed of individuals. An individual is a potential solution of the optimization problem (*i.e.,* the selection of the best features for detecting the running application). Individual can be scored using at least one fitness function (FF). In our study, the individual is composed of a vector of 1 and 0 indicating whether or

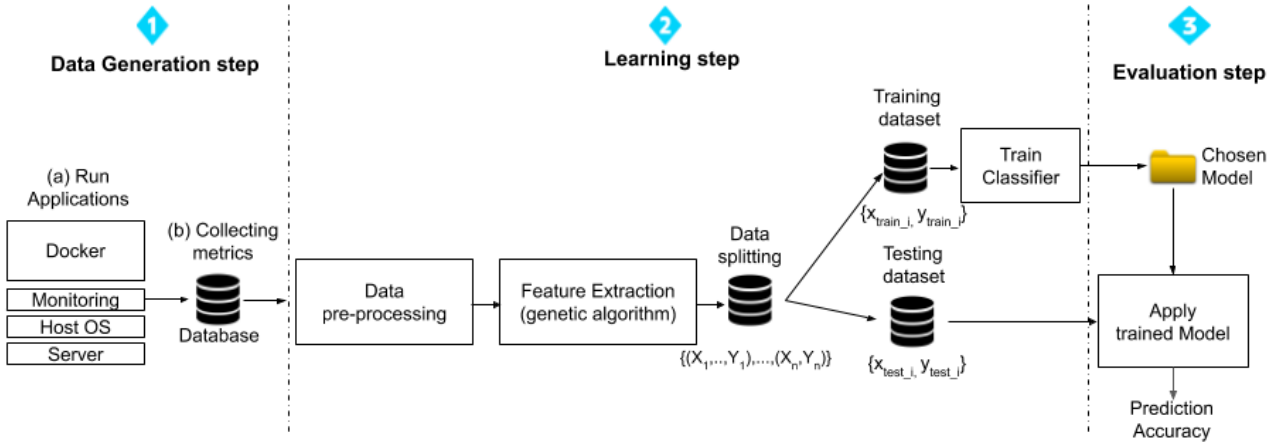

Figure 2. Training Fingerprint Models Approach

not the feature (*i.e.,* the metrics) is selected and the fitness function which is the accuracy classification score calculated by counting the number of correct classifications and divide it by the total number of samples. At each GA step, individuals from a generation of a population mutate using two-point crossover to generate new individuals who inherit from both parents on which random flip could be applied (*i.e.,* a previously selected feature can become unselected). Then, through the fitness function, the best children of the new generation (*i.e.,* those who maximize the classification score) are selected to produce the next generation. Finally, after 100 generations the selected features are then used in a classification process with a Random Forest classifier.

B. *Fingerprint Tracker: Tracking application executions*

This section details how the fingerprint tracker leverages the fingerprint recognition models in order to ascertain both remotely and online whether or not a sabotage is taking place.

In Figure 3 we highlight the interactions that occur between the trusted (*i.e.,* Operator) and the trustless environment (*i.e.,* Farmer) in order to track the correct execution of the customer applications. To do so, the methodology is the following. First, the operator requests the execution of an application to a farmer (*i.e.,* a trustless resource provider) (1), which subsequently triggers the creation of a container that will host the execution of the application. Then, this designated farmer is asked to supply, every second, the resource usage measurements of the machine that is used to execute the customer application within a predefined time interval (3). These measurements are then ingested by the *Fingerprint Tracker* to detect the correctness of the execution of the application (4). If either the resource usage measurements are not delivered in a timely fashion or the *Fingerprint Tracker* detects that the fingerprint is not complying with the expected one for a duration of at

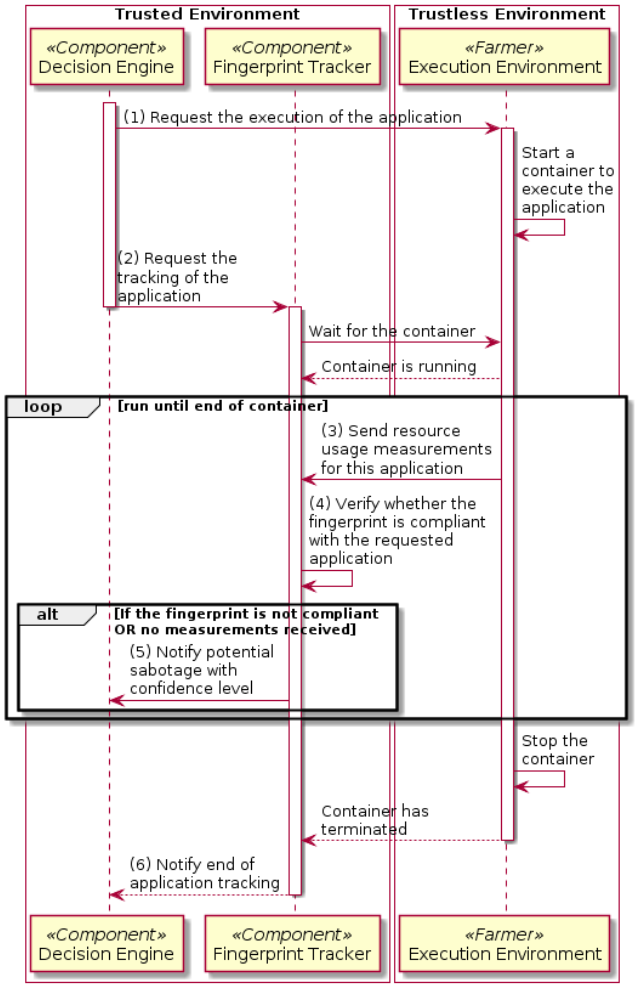

Figure 3. Sequence diagram of the interaction between the trusted environment and the trustless environment

least 2 minutes (this duration can be adjusted based on the desired confidence level as explained in Section IV), then the *Fingerprint Tracker* considers that there is a sufficiently high likelihood that a sabotage has taken place during this time frame. In such a case, the *Fingerprint Tracker* notifies the *Decision Engine* of a potential sabotage with the associated confidence level (5) so that it can trigger countermeasures, such as spot-checking with blacklist [7]. Finally, upon completion of the application and after its hosting container has stopped, the *Fingerprint Tracker* ends its tracking and notifies it to the decision engine (6).

IV. Evaluation

This section describes the obtained results. Through this experimental part, we try to answer 4 research questions (RQ):

- **RQ1**: What are the best features for tracking application fingerprint?
- **RQ2**: What is the accuracy of the fingerprint Tracker for the three use cases: *homogeneous*, *heterogeneous*, and *unspecified* hardware?
- **RQ3**: How does the accuracy change with regards to the size of the training dataset (learning curve)?
- **RQ4**: What is the minimum period of monitoring required?

A. Experimental setup

We used four heterogeneous physical machine configurations running Ubuntu 14.04 with Docker 18.06.0-ce. Table II describes the hardware characteristics of machines used by the farmers. We used two DELL server configurations that are very common in data center infrastructures and two uncommon configurations (a laptop and an embedded board).

We made use of Python with the *scikit-learn* [24] version 0.18 library which provides state-of-the-art machine learning algorithms. Besides, all training and forecasts done by the Operator were performed on servers with an Intel(R) Xeon(R) E5-2630 v2 CPU clocked at 2.60GHz and with 130GB of RAM. In our experiments, we used five applications: video processing, 3D rendering, email server, software development, and web application, which are detailed in Section III-A1. We used the Ubuntu 14.04 LTS GNU Linux distribution with a kernel version 4.2 for M1, M2, and M3, and for M4 a kernel 4.14. The virtualization system used was Docker version 18.06. Finally, we have experimented with 4 heterogeneous physical machines in terms of CPU performance and architecture, and storage (*e.g.,* SSD or HDD) to explore the fingerprint accuracy as compared to the used hardware.

Table II
Farmer physical Machines

ID	CPU	Memory (GB)	Storage
M1	Quad core Intel Core i7-4900MQ	15	Samsung Evo 850
M2	Hexa core Intel Xeon E5-2630	130	Intel Solid-State Drive 750
M3	Hexa core Intel Xeon E5-2630	130	Samsung 960 Pro
M4	ARM Cortex-A53	1	Kingston microSDHC

*1) **RQ1-Selected features**:* Our approach uses a genetic algorithm to select a subset of the monitored metrics to be used to efficiently train the fingerprint models. For the 5 applications, it emerged that among the 48 metrics the GA method has selected a total of 5 features for homogeneous hardware (see Table III) and 13 features for heterogeneous hardware (see Table IV) for all the applications. We observed that they are mainly related to CPU, memory and storage usage.

Table III
Selected Features for Homogeneous Hardware

Name	Description
active-anon	Anonymous memory that has been used more recently
pgpgin	Number of kilobytes the system has paged in from disk per second.
I/O write and sync	Number of I/O operations
write-bytes	Bytes written per second to disk

Table IV
Selected Features for Heterogeneous Hardware and Unspecified Hardware

Name	Description
cpu-usage	Percentage of CPU utilization
active-anon	Anonymous memory that has been used more recently
inactive-anon	Bytes of anonymous and swap cache memory on inactive LRU list
pgpgin	Number of kilobytes the system has paged in from disk per second.
pgfault	Number of page faults the system has made per second
active-file	Bytes of file-backed memory on active LRU list.
I/O read, write and sync	Number of I/O operations
mapped-file	Bytes of mapped file (includes tmpfs/shmem)
read-bytes	Bytes read per second from disk
write-bytes	Bytes written per second to disk
writeback	Bytes of file/anon cache that are queued for syncing to disk.

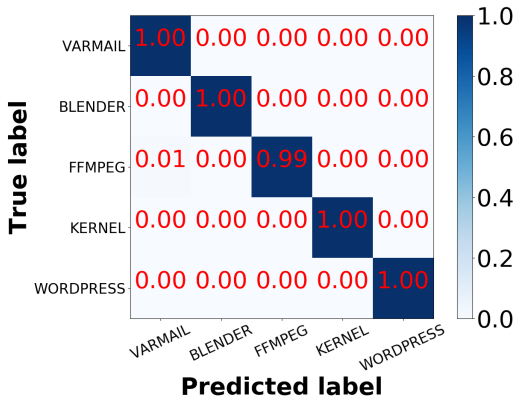

Figure 4. Confusion Matrix with Homogeneous Hardware

*2) **RQ2-Accuracy**:* The confusion matrix shown in Figure 4 was built as follows: we ran each application 50 times by randomly selecting each time 70% of the dataset

(comprising all the applications) to build the model and the remaining 30% to evaluate its accuracy for a given hardware architecture. For each execution, we have fixed the hardware architectures and assumed that on the trustless side, the hardware used was the same (*i.e.,* the homogeneous hardware case). We observed that the resulting predictive fingerprint recognition model was very accurate and succeeded in distinguishing between the 5 applications with an accuracy of 99.88%.

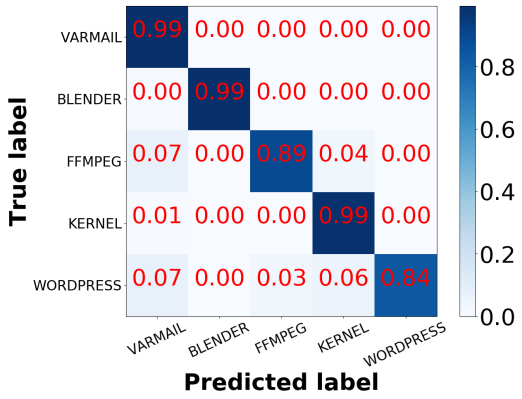

Figure 5. Confusion Matrix with Heterogeneous Hardware

Figure 5 follows the same methodology as the previous experiment but for the *heterogeneous hardware* case (*i.e.,* four hardware architectures were combined in a unique dataset so that an application could be run on very different hardware on the trustless side). We remark that WordPress was the most inaccurate application to track with an accuracy of 84%.

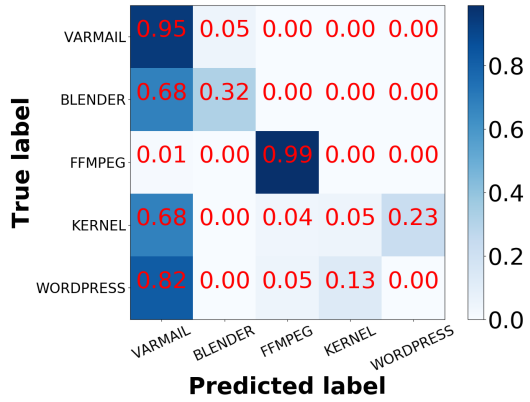

Figure 6. Confusion Matrix on Unspecified Hardware

Figure 6 shows the confusion matrix for five applications in case of *unspecified hardware*. The accuracy is evaluated as follows: during the training the hardware M1, M2 and M3 are used and then M4 is used for testing. We have chosen the M4 hardware for the test because it is the most different in terms of hardware characteristics. The goal is to be able to evaluate the impact on the fingerprint recognizer accuracy in the (extreme) unspecified hardware case. We observe that compared to the heterogeneous hardware case, the accuracy drops to about 40%. This result means that the application fingerprinting technique may not be relevant when the hardware used for the training is too different from the one used for the test.

*3) **RQ3-Learning curve**:* The learning curve shows the evolution of the model accuracy according to the number of training samples [10], [25]. In order to build our learning curve, we performed a progressive sampling by increasing the dataset sizes $N_{training}$ = 1 to N_{max} with a step of 1 second. N_{max} is the total number of samples available.

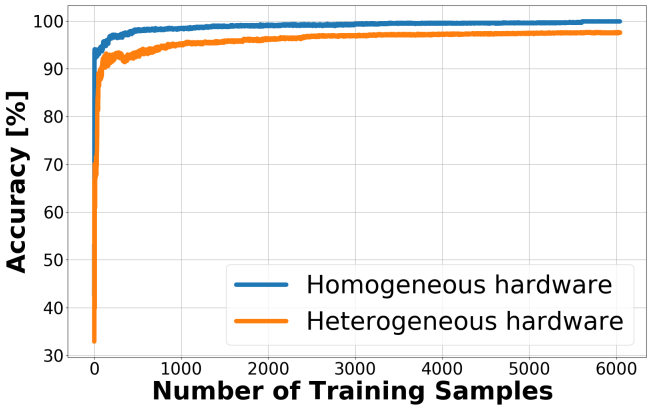

Figure 7. Learning curves on the testing set as a function of the number of training samples

In Figure 7 we show the accuracy of the algorithms according to the training set size. First, we observe, as expected, that the accuracy improves with the increase of the training set size. In case of homogeneous hardware, we observe that with 3000 samples (*i.e,* 5 minutes) the accuracy reaches 99.95% and 100% with 5500 samples. As compare to heterogeneous hardware, we notice that we need about 3600 samples (*i.e.,* 60 minutes) of application trace to be able to distinguish between the 5 five applications with an accuracy of 97% and it reaches 98% with 5700 samples. Moreover, after about 100 minutes the accuracy does not increase anymore.

*4) **RQ4-Monitoring Interval**:* In Figure 8, we show the size of the monitoring interval in seconds used to predict the running application with regards to the accuracy for the three use cases: homogeneous, heterogeneous and unspecified. We observe that only 1 second is required to achieve 100% accuracy in case of homogeneous hardware. In contrast, we notice that 60 seconds are needed to reach an accuracy of 98% for heterogeneous hardware. Finally, in the unspecified hardware, we observe that the accuracy is capped to 40% and does not improve after about 40 seconds.

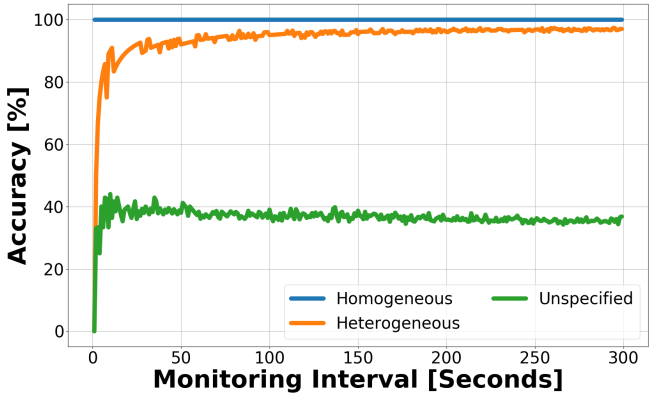

Figure 8. Accuracy curves as function of the number of testing samples

V. Discussions

This application fingerprinting mechanism based on consumed resources can be considered as a second level of security that can be coupled with other existing approaches already proposed in the state of the art studies. Indeed, it may be difficult for a malicious user to cheat on both the output format of the application and the associated stream of measurements that lead to this incorrect result.

In addition, to prevent a saboteur from saving the application usages (*e.g.,* cpu usage) and then sending these metrics to the operator, three actions could be implemented:

1) First, the decision engine scheduler may try to avoid to schedule the same application to the same farmer several times.
2) Second, we propose to use the technique of *Proof of Storage* [26] to ensures that the data is actually stored in the trustless environment.
3) Third, the application binary could be obfuscated to prevent potential reverse engineering by analyzing the binary.

The poor accuracy of fingerprint recognizer in the *unspecified hardware* case is not a surprise. Indeed, for example, when an application is executed on a physical machine with a large volume of memory buffer, the operating system may delay disk write operations and thus improve the performance [27]. This could prevent the ML algorithm from identifying the relationship between the application and the selected feature metrics (*e.g.,* utilization of the memory, write back). A consequence, it may fail in identifying the application.

There are several parameters that could affect the accuracy of the model. In the performed experiments, we did not test the sensitivity of the model to changes in kernel parameters. It may be relevant to evaluate the kernel configuration parameters that have a significant impact on model accuracy. Container resource allocation can be configured at runtime using the CGroup configuration interface [2]. As with the static configuration parameter of the kernel, it may be relevant to assess the impact of such a dynamic evolution on the model accuracy. In addition, the virtualization technique used such as Docker, Lxd, Qemu could also have an impact on the model accuracy.

Moreover, the use of a different version of the same application or the modification of the application parameters may also induce a change in the estimated model due to a change in its behavior. We did not consider how co-located application workloads may interfere on applications. We have considered only a fixed capacity per host, while the capacity may depend on all running applications [28]

Finally, monitoring may also affect the estimated model. Indeed, a modification of the implementation of the monitoring component could also affect the model accuracy (*e.g.,* the CPU sampling differs between the training and the testing phase).

VI. Related work

Numerous studies have already evaluated the correctness of an execution in a trustless environment. In this section, we first focus on the work that tries to validate the computation results. Then, we discuss studies that try to ascertain the correct execution of applications.

A. Ensuring the result correctness and detecting sabotage

In [7], the objective is to execute the same work unit N times before eventually comparing them - each result being a vote - until converging towards a result. This method has the advantage of ensuring a very high level of certainty with regards to the correctness of the result of a given work unit. However, this comes with two major drawbacks. First, it has a high overhead: N times the initial execution cost with N being the number of votes. Second, the time this method requires to ascertain the falseness of a given execution can be excessively long as many rounds of voting may occur, postponing the decision every time.

In [29], [30], a different strategy was used. Contrary to the previous method, the goal here is to minimize the overhead of checking the correctness of an execution by submitting the various resource providers to test. The kind of tests that can be used may rely on four different techniques: naive, quiz, ringers, and spot checking (see [7], [31]).

B. Application identification and detection

The fingerprinting approach tries to generalize application behavior given its execution traces into a model. This approach has been used in different contexts to identify and detect applications. In [32], [33], the authors show the benefits of fingerprints to automatically detect hardware Trojan and in [34], Lin *et al.* are using packet size distribution of the connections to create an application fingerprint.

[2]https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt

Side-channel analysis is composed of two steps, commonly referred to identification and exploitation. The identification consists in understanding the leakage and building suitable models. The exploitation consists of using the identified leakage models to extract an information. To build the model, several approaches have shown that it can be approximated in a profiling phase using machine learning techniques. In [35], Gulmezoglu *et al.* show that the use of cache access profiles could be efficient to classify applications. Zender *et al.* [36] show the efficiency of unsupervised machine learning to automatically classify traffic and application when the network traffic. In [37], Schuster *et al.* show that by monitoring an encrypted network traffic a convolutional neural networks can accurately distinguish movies streamed using network traffic bursts.

In this paper, we use machine learning for side-channel analysis. We mainly demonstrate on a set of real applications that a set of 48 metrics commonly used in cloud technology could be utilized to classify application without any assumptions on applications. Contrary to existing work, we neither do any assumption on these metrics nor on the resource usage type of the application (*e.g.,* CPU or memory Intensive) to classify.

VII. Conclusion

Tracking the correctness of the application execution over time is necessary to prevent malicious infrastructure owners from sabotaging the computation. Machine learning combined with a fingerprint technique seems to be a relevant approach for homogeneous and heterogeneous hardware. It also shows that the approach is not viable for unspecified hardware.

This paper shows that it is not necessary to take into account application characteristics when trying to track the execution of applications when using our fingerprinting approach that combines both a genetic and a machine learning algorithm.

We evaluated our approach with RF. Our results show that we were able to detect the correct execution with an accuracy of 99.88% with homogeneous hardware, 98% with heterogeneous and 40% with unspecified hardware on the five selected applications.

We will also work toward considering GPU and other processing elements. We also plan to evaluate deep learning algorithms such as Long Short Term Memory (LSTM) which is designed to capture dependencies within an input sequence and GAN (Generative Adversarial Network). Finally, other experiments are underway to apply one-class (*i.e.,* unary classification) to train a model per application.

Finally, as for future work, the prediction ability for *unspecified hardware* could be improved by normalizing the performance metrics using hardware information provided for example by sysconf. In addition, during the feature selection step, we could also add in the testing set *unspecified hardware* to let the feature selection algorithm select feature more robust to hardware variations.

Acknowledgment

This work was supported by the Institute of Research and Technology b-com, dedicated to digital technologies, funded by the French government through the ANR Investment referenced ANR-A0-AIRT-07.

References

[1] S. Kaisler, F. Armour, J. A. Espinosa, and W. Money, "Big data: Issues and challenges moving forward," in *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pp. 995–1004, IEEE, 2013.

[2] J.-E. Dartois, A. Knefati, J. Boukhobza, and O. Barais, "Using quantile regression for reclaiming unused cloud resources while achieving sla," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 89–98, IEEE, 2018.

[3] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@ home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[4] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor-a hunter of idle workstations," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 1987.

[5] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp. 124–134, ACM, 2011.

[6] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure linux containers with intel SGX," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, (Savannah, GA), pp. 689–703, USENIX Association, 2016.

[7] L. F. Sarmenta, "Sabotage-tolerance mechanisms for volunteer computing systems," *Future Generation Computer Systems*, vol. 18, no. 4, pp. 561–572, 2002.

[8] "cadvisor online documentation." Website, 2019. Accessed May, 27st, 2019.

[9] G. Aceto, A. Botta, W. De Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, no. 9, pp. 2093–2115, 2013.

[10] E. Alpaydin, *Introduction to machine learning*. In MIT press, 2014.

[11] J.-E. Dartois, H. B. Ribeiro, J. Boukhobza, and O. Barais, "Cuckoo: a mechanism for exploiting ephemeral and heterogeneous cloud resource," in *IEEE International Conference on Cloud Computing*, IEEE, 2019.

[12] W. Reese, "Nginx: the high-performance web server and reverse proxy," *Linux Journal*, vol. 2008, no. 173, p. 2, 2008.

[13] A. MySQL, "Mysql," 2001. https://www.mysql.com.

[14] A. Brazell, *WordPress Bible*, vol. 726. John Wiley and Sons, 2011.

[15] V. Tarasov, E. Zadok, and S. Shepler, "Filebench: A flexible framework for file system benchmarking," *The USENIX Magazine*, vol. 41, no. 1, 2016.

[16] F. Bellard, M. Niedermayer, *et al.*, "Ffmpeg," *Available from: http://ffmpeg.org*, 2012.

[17] R. M. Stallman *et al.*, "Using the gnu compiler collection," *Free Software Foundation*, vol. 4, no. 02, 2003.

[18] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*, pp. 2962–2970, 2015.

[19] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[20] A. Jain and D. Zongker, "Feature selection: Evaluation, application, and small sample performance," *IEEE transactions on pattern analysis and machine intelligence*, vol. 19, no. 2, pp. 153–158, 1997.

[21] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[22] H. Hao, C.-L. Liu, and H. Sako, "Comparison of genetic algorithm and sequential search methods for classifier subset selection," in *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pp. 765–769, Citeseer, 2003.

[23] J. Yang and V. Honavar, "Feature subset selection using a genetic algorithm," in *Feature extraction, construction and selection*, pp. 117–136, Springer, 1998.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[25] T. H. R. T. J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics, 2013.

[26] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 491–500, Acm, 2011.

[27] R. Karedla, J. S. Love, and B. G. Wherry, "Caching strategies to improve disk system performance," *Computer*, vol. 27, no. 3, pp. 38–46, 1994.

[28] J.-E. Dartois, J. Boukhobza, A. Knefati, and O. Barais, "Investigating machine learning algorithms for modeling ssd i/o performance for container-based virtualization," *IEEE Transactions on Cloud Computing*, vol. 14, pp. 1–14, 2019.

[29] W. Du, M. Murugesan, and J. Jia, "Uncheatable grid computing," in *Algorithms and theory of computation handbook*, pp. 30–30, Chapman & Hall/CRC, 2010.

[30] S. Zhao, V. Lo, and C. G. Dickey, "Result verification and trust-based scheduling in peer-to-peer grids," in *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, pp. 31–38, IEEE, 2005.

[31] P. Golle and I. Mironov, "Uncheatable distributed computations," *Topics in CryptologyCT-RSA 2001*, pp. 425–440, 2001.

[32] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using ic fingerprinting," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, pp. 296–310, May 2007.

[33] Yier Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, pp. 51–57, June 2008.

[34] Y.-D. Lin, C.-N. Lu, Y.-C. Lai, W.-H. Peng, and P.-C. Lin, "Application classification using packet size distribution and port association," *Journal of Network and Computer Applications*, vol. 32, no. 5, pp. 1023 – 1030, 2009. Next Generation Content Networks.

[35] B. Gulmezoglu, T. Eisenbarth, and B. Sunar, "Cache-based application detection in the cloud using machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, (New York, NY, USA), pp. 288–300, ACM, 2017.

[36] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)l*, pp. 250–257, Nov 2005.

[37] R. Schuster, V. Shmatikov, and E. Tromer, "Beauty and the burst: Remote identification of encrypted video streams," in *26th USENIX Security Symposium (USENIX Security 17)*, (Vancouver, BC), pp. 1357–1374, USENIX Association, 2017.